

# Git e Github - Iniciante

quarta-feira, 24 de novembro de 2021 17:43

# Introdução - 1º Aula

quinta-feira, 25 de novembro de 2021 15:02

Vinicius Dias - Guiarei vocês neste curso de **Git e Github**

**Git -> é um Sistema de controle de versões**

**Existem outros como o GIT.. CVS, SVN, Mercurial.**

**Instalação do git: [git-scm.com](https://git-scm.com)**

## O que aprendi:

- O que são (e para que servem) **sistemas de controle de versões** e como eles podem ajudar o nosso fluxo de desenvolvimento
  - Nos ajudam a manter um histórico de alterações;
  - Nos ajudam a ter controle sobre cada alteração no código;
  - Nos ajudam para que uma alteração de determinada pessoa não influencie na alteração realizada por outra;
  - Etc.
- O que é o Git e como instalá-lo
- Que com o comando `git init` nós conseguimos criar um repositório Git;
- Como analisar o estado do nosso repositório através do comando `git status`.

`<type>[optional scope]: <description>`

## Alguma configurações:

- `Git config --local user.name <seu nome>` -> define o nome do repositório local
- `Git config --local user.email <seu email>` -> define o email do repositório local
- `Git config --global core.pager cat` -> define a leitura do log, ao invés de pages ele irá somente imprimir no terminal.

# Salvando alterações - 2º Aula

quinta-feira, 25 de novembro de 2021 15:00

- `git add ->` Para adicionar o monitoramento do git
- `Git restore --staged <file> ou . ->` Retira o monitoramento para o estado anterior
- `git commit -m "Mensagem" ->` Para criar um checkpoint em cima das alterações feitas.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Cursos de DevOps da Alura</title>
</head>
<body>
  <ul>
    <li>Vagrant: O curso que mudara sua estrutura</li>
    <li>Docker: Criando containers sem dor de cabeça</li>
    <li>Ansible: Infraestrutura como código</li>
    <li>Integração Contínua!</li>
    <li>Café em casa</li>
  </ul>
</body>
</html>
```

O comando `add` ele serve para que o git comece a enxergar aquele arquivo, fazendo com que coloque estes arquivos na área de stage, pronto pra ser commitado quando mandarem. Ou seja, eles estão preparados.

- `git log ->` que mostra o histórico de alterações no prompt de comando.
- `Git log --oneline ->` exibir o log mais resumido.
- `Git log -n <valor> ->` mostrar somente uma quantidade de commits, sendo 1 = o ultimo commit feito
- `.gitignore ->` para dizer qual arquivo ou pasta ele irá ignorar completamente.

# Repositórios remotos - 3º Aula

quinta-feira, 25 de novembro de 2021 18:07

- `git init --bare` -> pra você inicializar o repositório, dizendo que ele é só para armazenar alterações.
  - `git remote add local "URL ou LINK"` do repositório aonde será feito o fetch e o push dos arquivos.
  - `Git clone "URL ou LINK"` do repositório que quer clonar.
  - `Git push nome do remote e nome da branch` -> para empurrar os arquivos commitados no repositório.
  - `Git pull nome do remote e nome da branch` -> para puxar os arquivos do repositório.
- **CRIAÇÃO da conta no gitHUB -> Compartilhando os dados pela internet.**

# Branches - 4º Aula

sexta-feira, 26 de novembro de 2021

11:45

- Git branch (nome da branch) -> **criar** uma nova branch
- Git checkout -b (nome da branch) -> **criar e entrar** na branch
- Git checkout (nome da branch) -> **navegar** entre as branches
- Git merge (nome da branch) -> **cria um novo commit** unindo duas linhas de produção do git
- Git rebase titulo -> coloca os commits em uma só linha de produção.

Evitando maiores conflitos, é de boa prática que antes de começar a trabalhar você utilize o **pull** para verificar se já não tem alterações feitas e atualizar o seu repositório.

# Ctrl + Z no Git - 5ª Aula

sexta-feira, 26 de novembro de 2021 16:17

## Ctrl + Z no Git:

- Git checkout -- <file> -> remove alterações recentes
- Git reset HEAD <file> -> remove da área de stage
- Git restore <file> -> remove alterações recentes
- Git restore --staged <file> -> remove da área de stage
- Git revert <hash> -> remover um commit -> deixando no histórico o rastro do revert

## Guardando para depois:

- Git stash -> esconder as modificações feitas, revertendo-as e elas ficaram esperando salvas na área de WIP (working in progress).
- Git stash list -> lista de todos os stashes que estão aguardando..
- Git stash apply <numero da stash> -> pega a stash novamente.
- Git stash drop <numero da stash> -> remove a stash da area do WIP
- Git stash pop <numero da stash> -> pega a stash e já remove da area do WIP

## Viajando no tempo:

- Git checkout <hash> -> navega para o ponto de um commit anterior.

# Vendo as alterações - 6º Aula

sexta-feira, 26 de novembro de 2021

19:46

## Vendo as alterações:

- `Git diff` -> revela as mudanças feitas no código
- `Git diff <hash 1 commit ou branch>..<hash 2 commit ou branch>` -> compara dois commits

## Tags e releases:

- `Git tag -a <nome da tag> -m <comentario>` -> adiciona um marco no ponto que quiser.
- `Git tag` -> mostra as tags existentes.
- Da para enviar com o `git push` somente a tag.

# Git e Github: Estratégias de ramificação, Conflitos e Pull Requests



# Github e Open Souce - 1º Aula

segunda-feira, 29 de novembro de 2021 10:24

- Veremos o que é uma issue
- gerenciar e lidar com pull request
- Gestão de conflitos
- Busca avançada de commits
- Estratégias de branching
- Conhecendo ferramentas como GitKraken
- Conhecendo eventos como git hooks

# Controle avançado de conflitos - 2º Aula

segunda-feira, 29 de novembro de 2021 10:36

## Issues:

Sugestões de melhorias, pedidos de novas funcionalidades, e assim por diante.

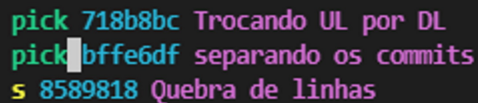
## Pull Requests:

Pedido de envio para o repositório de alguém, enviando certas modificações/alterações em um código.

Você pode fazer um Fork de um projeto, realizar suas modificações e abrir um Issue e enviar suas alterações por PR para resolver esta Issue.

## Unindo commits, para facilitar a revisão do PR:

- Git rebase -i HEAD~3
- Ou
- Git rebase -i <commit anterior daquele que vc quer juntar>



```
pick 718b8bc Trocando UL por DL
pick bffe6df separando os commits
s 8589818 Quebra de linhas
```

- Trocar o <pick> por <s> de squashing = esmagamento.

## Alternativas ao GitHub:

- GitLab (ótima para project corporation)
- Bitbucket (ótimas free actions)

# Pegando um commit - 3º Aula

segunda-feira, 29 de novembro de 2021 13:42

## Pegando um commit:

- Git cherry-pick <nome do commit> -> buscar um commit específico para uma branch.

## Encontrando bugs: bisect

- Git bisect start -> iniciando a busca

Depois de iniciada, é preciso falar qual o ponto em que deseja iniciar aonde o estado do código não está boa.

- Git bisect bad HEAD -> estado atual RUIM..

Depois informamos o estado que possivelmente estaria bom.

- Git bisect good <hash do commit> -> Estado possivelmente bom.

Com isso você define o range da busca.

- Git bisect good -> quando o commit que você procurava já foi encontrado
- Git bisect bad -> quando ainda não encontrou o que procurava.
- Git show <hash do commit> -> para ver determinado commit

## "Encontrando o culpado (blame)":

- Git blame <arquivo> -> Consegue enxergar quem realizou todas modificações de cada linha pelo comando do git.

# Estratégias de branching - 4º Aula

segunda-feira, 29 de novembro de 2021 14:40

## Master e produção:

- Uma convenção aonde a master é a linha principal aonde inicia e finaliza o projeto.

## Mais branches:

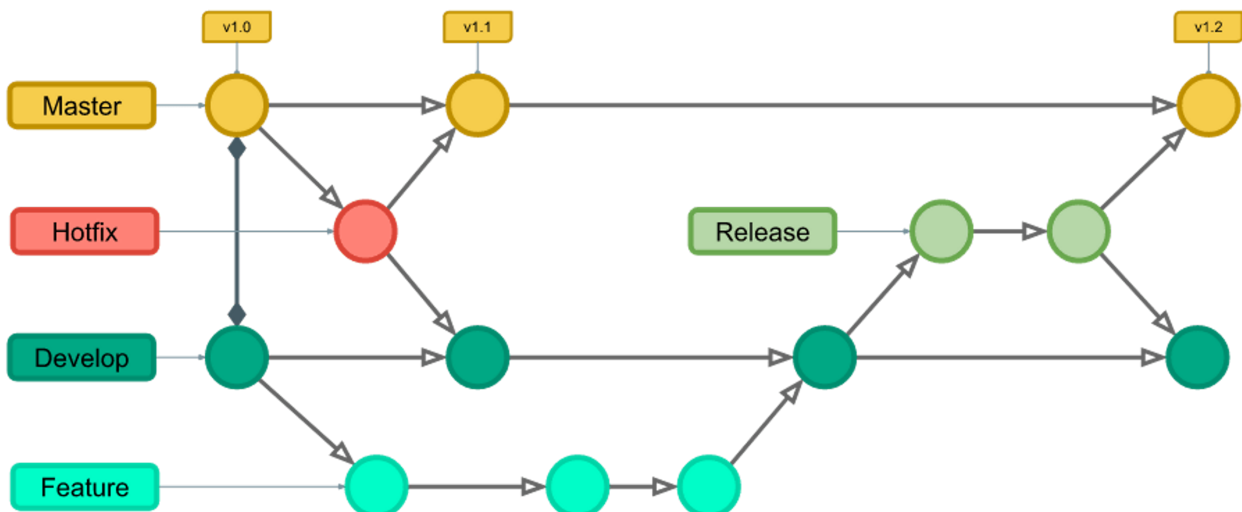
➤ Nunca mexa na Master (Produção), pois ela normalmente é a branch de produção.

- `Git branch -d <nome da branch>` -> para deletar uma branch
- `Git branch -D <nome da branch>` -> para forçar o delete caso a branch esteja a frente da master

## Branches: (Git Flow)

- Development -> Branch com o estado mais próximo da master.
- Features/.. -> São as subs development branches, aonde são feitas todas evoluções do código, somando todas Features da a Branch de Development.
- Release/.. -> Branch para serem testadas as features, para assim ser mandada para a Development.
- Hotfix/.. -> para arrumar bugs em produção, a partir da master e para o development que tem que estar sempre atualizada.

## Git Flow:



# Ferramentas Visuais - 5ª Aula

segunda-feira, 29 de novembro de 2021 16:45

**Git Cola:** Ferramenta que te dá uma aplicação mais visual dos comandos do git. **OBS:** (Não achei muito legal, mas tá aí kkkk)

**Github desktop:** Ferramenta do próprio gitHub que também dá uma facilidade para os comandos git.

**GitKraken:** Ferramenta que ajuda nos comandos do git, e também ajuda a trabalhar com o git Flow, facilitando também com a questão visual. **OBS:** (Achei muito bom!)

# Hooks e deploy com Git - 6º Aula

segunda-feira, 29 de novembro de 2021 17:15

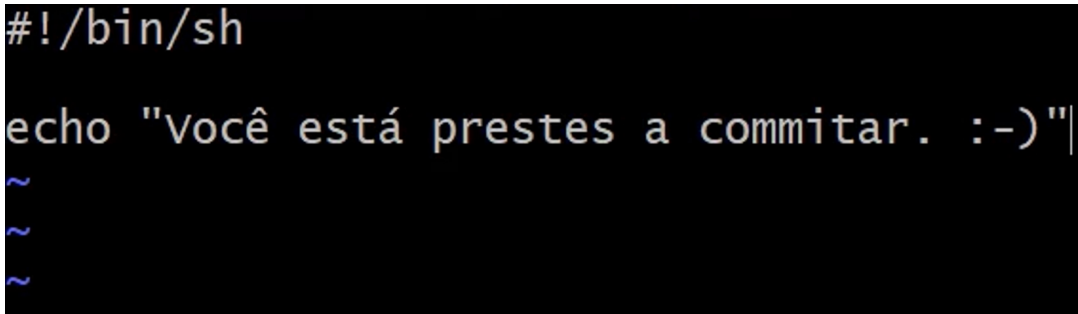
## Eventos no Git:

Acessando a pasta .git que é criada quando você tem um servidor de git, você encontra a pasta Hooks.

La dentro você tem vários sample para realizar algum script quando alguém fazer alguma coisa, como por exemplo, commitar.

Dentro da pasta hooks, digite:

- Vim pre-commit -> para abrir o arquivo usando o editor vim



```
#!/bin/sh
echo "Você está prestes a commitar. :-)"
~
~
~
```

- 
- O comando echo, ele reproduz no terminal aquilo que você quiser digitar.
- Digitando ESC - ':' - X -> Você sai do editor.
- Cat <nome do arquivo> -> mostra o que tem dentro do arquivo.

## <Post receive> -> Quando fazer um post..

Git --git-dir="pasta do repositório - servidor" --work-tree="pasta aonde está o projeto - web" checkout -f -> criando uma pasta web para carregar os arquivos lá dentro do servidor local.